Design Project

Design Report

16 April 2021

Authors:

Weiting Cai s2067684

Di Zhuang s2131080

Ruilin Yang s2099497

Supervisor:

Tom van Dijk

Table of Content

Table of Content 1

1. Introduction 4

2. Requirement specification 5
- 2.1. Functional requirements 5
- 2.2. Non-functional requirements 5

3. Global design 7
- 3.1. Graph visualization library 7
- 3.2. Programming language 8
- 3.3. Architecture 8
  - 3.3.1. Backend 8
  - 3.3.2. Protocol between backend and frontend 9
  - 3.3.3. Frontend 9
    - 3.3.3.1. Visibility of system status 9
    - 3.3.3.2. Match between system and the real world 10
    - 3.3.3.3. User control and freedom 10
    - 3.3.3.4. Error prevention & recovery 11
    - 3.3.3.5. Aesthetic and minimalist design 11

4. A detailed design choices with justifications 12
- 4.1. Labels of a node 12
  - 4.1.1. Overall design of label 12
    - 4.1.1.1. labels can displayed as either text or color 12
    - 4.1.1.2. Color effect of the closest parent of a node 12
  - 4.1.2. Visualization of label 13
    - 4.1.2.1. Fetch the labels of selected algorithm 13
    - 4.1.2.2. Display of color label 13
    - 4.1.2.3. Display of text label 13
- 4.2. Frontend 14
  - 4.2.1. Collapsible Sidebar 14
  - 4.2.2. List of steps 14
  - 4.2.3. Button names and colors 14
  - 4.2.4. Sequence of options in the collapsible sidebar 14
- 4.3. Backend 14
  - 4.3.1. The data format to describe a node / a game 14
  - 4.3.2. The choice of birdview game status 15
  - 4.3.3. Return steps to the frontend in one shot 15

5. Testing 16

6. Overview of the source code **17**
6.1. Frontend 17
6.2. Backend 18
6.2.1. Package resources 18
6.2.2. Package algorithms 19
6.2.3. Package control 19
6.2.4. Package parser 19
6.2.5. Package modelGame 19
6.2.6. Package modelStep 20
6.2.7. Package test 21

7. User manual **22**
7.1. Compatibilities and Installation 22
7.1.1. Backend compatibility 22
7.1.2. Frontend compatibility 22
7.1.3. Installation 22
Step 1. Install Tomcat and Maven: 22
Step 2. Create a user in Apache Tomcat: 23
Step 3. Create a user in Apache Maven: 23
Step 4. Start Tomcat: 23
Step 5. Deploy the application: 24
7.2. Frontend Usage 24
7.2.1. Add Parity Game 24
7.2.1.1. Create parity game by hand 24
7.2.1.2. Import parity game from local .pg file 25
7.2.2. Configure Parity Game 27
7.2.2.1. Add node/edge 27
7.2.2.2. Set priority 27
7.2.2.3. Increment/Decrement priority 28
7.2.2.4. Toggle ownership 28
7.2.2.5. Delete node/edge 29
7.2.2.6. Copy/paste subgraph 29
7.2.3. Algorithm 30
7.2.3.1. Run algorithm 30
7.2.3.2. Stop algorithm 33
7.2.4. Export Game/Solution 34
7.3. Add customized algorithm 34
7.3.1. Procedure 34
7.3.2. Note 35

8. Individual contribution **37**
8.1. Weiting 37

8.2. Di 37
8.3. Ruilin 37
8.4. Group work 38

9. Future work 38

References 38

Appendix I: requirement specification 39

Appendix II: test plan and test result 43
Unit testing 43
Big-Bang Integration Testing 47
Choose algorithm & game 47
After running the algorithm 47
Export and clear are parallel to the above 48

Appendix III: weekly meeting/demo with supervisor 49
Week 1 - Feb 4, Thu 49
Week 2 - Feb 11, Thu 49
Week 3 - Feb 18, Thu 49
Week 4 - Mar 1, Mon 49
Week 5 - Mar 8, Mon 50
Week 6 - Mar 15, Mon 50
Week 7 - Mar 22, Mon 50
Week 8 - Mar 29, Mon 51
Week 9 - Apr 6, Tue 51
Week 10 - Apr 12, Mon 51

Appendix IV: sprint plan and result 52

1. Introduction

This project is aimed at building a GUI interface that helps with parity game algorithm research. In the research process, one high-frequency operation is laborious and error-prone, that is to adjust the configuration of a parity game. Also, it is inconvenient that the researcher cannot see a visual reflection of the steps of the algorithms.

The solution to this problem is a GUI that can automate the editing process by replacing editing text in files with a more intuitive way, display the steps so that the researchers can gain better understanding of their algorithms, and also capable of doing other peripheral tasks such as import from standard parity game format and export to such format.

The stakeholder is the research community on parity game algorithms, which is represented by the supervisor of this project, Tom van Dijk.

In this report, several aspects of the project are illustrated. In particular, in Chapter 2 and Appendix I, the functional requirements and the non-functional requirements are specified; in Chapter 3 and 4 the global and detailed design choices are discussed and justified; in Chapter 5 and Appendix II the settings of the unit testing and the system testing are explained and the results of the testing are summarized. Chapter 6 provides the maintainter with an overview of the source code, and subsequently in Chapter 7 a user manual is provided. Chapter 8 is a reflection over team collaboration. Finally, a brief discussion over possible future work is given in Chapter 9.

2. Requirement specification

The project team first mined the possible requirements from the project description, and then got the input from the client, including whether a specific requirement is valid, and if so, what priority should it have.

The requirement specification is adjusted across the first few weeks as the developers gained more understanding of the context, and as the client saw the prototypes to give feedback. The final version of requirement specification, including their priorities and whether they are implemented is in Appendix I. In this section there is a brief overview of the requirements.

2.1. Functional requirements

The functional requirements falls into a few categories:

Adjust the game	To edit the game in the GUI. This includes various operations such as adding or removing a node/edge, changing the properties of a single node or for a group of nodes. To facilitate the creation of a parity game, the nodes can be group selected and copied, they can also be deleted. And an undo function to recover from mis-operations.
Import/export	To import parity games from file, and export the game/solution of a game.
Visual aid for the research	Requirements about how the product can help researchers see algorithms visually.
Control	Requirements about start/stop/stepping of an algorithm.
Algorithms	Requirements about adding customized algorithms and built-in algorithms.

2.2. Non-functional requirements

There are only two non-functional requirements:

One is there should be an onboarding guide or quick help function in the GUI, this is replaced by a detailed user manual and a demo video (both information in chapter “User manual”).

The other is about the extensibility of the product. The client wants it to be extensible to other graph algorithm studies. The architecture of the product allows extension, but there could be a lot of work in the frontend, as in the frontend all functionalities are specific to the parity game, as developers we are not sure what are the commonalities between parity game and other graph research.

These two functionalities are also in the table in Appendix I.

3. Global design

This project aims to automate the manual work in parity game algorithm research to help the researchers focus on the algorithm itself, especially the laborious way of editing a parity game by altering the numbers of the text format of the game in a text editor. This means the user should be able to manipulate a parity game graph extensively in an intuitive way via the product.

A second goal of the project is to provide a visual aid to the researcher for gaining a better understanding of the algorithms. This means the appearance of a game graph should be updated in many aspects dynamically.
The choice of a visualization library is critical, as it is expected that a library that allows high interactivity and wide visualization possibility specialized in graphs is rare. So this choice is the starting point of the global design.

3.1. Graph visualization library

In making the choice, high interactivity, rich options in visualization, and specialization in graphs are the three determining characteristics. The chosen library for graph visualization is cytoscape.js (*Cytoscape.js*, n.d.), it is an open-source graph theory/network library written in Javascript. This decision is made in an explorative procedure.

First, research about suitable libraries is conducted. The vast majority of the visualization libraries/frameworks falls in one of the following categories:

- Not specialized in graph visualization, examples are D3.js (Javascript) and Matplotlib (Python). They are excellent at visualizing statistical data, but this project needs a library that can deal extensively with graphs.
- Specialized in graph visualization but can only create a static graph, examples are NetworkX (Python), igraph (Python), and graph-tool (Python), they focus more on the statistical analysis of graph-structured data, but not visualization of interactive graph.
- Can create dynamic graphs but the user has very limited interaction with it, for example, GraphStream (Java), the graphs can change dynamically, but can only be controlled by code.
- Can create a dynamic graph that users can interact with, but the options for developers are too primitive, such that it would be impossible for the developers to handle all the complexities in visualization in 10 weeks. JavaFX (Java) belongs to this category.

The research is of course not exhaustive, but it provides the landscape on the market: a suitable library is rare.

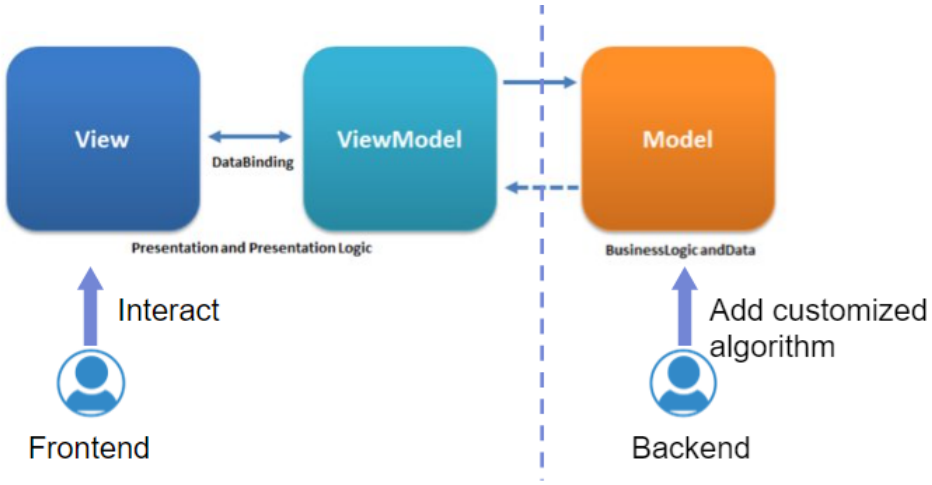
A second try is to narrow down the focus to libraries in Javascript, a language famous for its capabilities in building interactive apps, and choose the ones specialized in graph visualization. Several options are discovered and ruled out, for example, dracula.js and sigma.js for lack of detailed document, and vis.js for a less modern appearance.

The final choice of visualization library is cytoscape.js, it's funded by some US government institutions, developed by several research institutions and big companies, and has very sufficient documentation.

3.2. Programming language

The common programming languages shared across all the developers are Python and Java. Since a javascript library is chosen for visualization, it's natural to associate the product idea with the webapp that all developers learned in the first year at UT. To use the same tech stack, Java is chosen to build the backend.

3.3. Architecture



As it is decided to build a webapp, the Model-View-ViewModel (MVVM) architecture is especially suitable for this project: cytoscape.js can handle the binding of View and ViewModel; the researcher's algorithm is located in Model, it processes the data from ViewModel, and return data that contains information to modify ViewModel; finally, the changes in ViewModel is immediately reflected on the View.

With MVVM in mind, three pillars of the architecture will be introduced below.

3.3.1. Backend

The most important functionality in the backend is that the user should be able to add customized algorithms for research, and can select them in the frontend to run, as shown in the above illustration. The customized algorithms share some common functionalities, for example, they can all solve a parity game. Which algorithm object to instantiate is determined by the user at the frontend while the webapp is running.

The factory design pattern is natural for this purpose: it instantiates objects that implement a common interface in runtime.

3.3.2. Protocol between backend and frontend

From ViewModel to Model
It's provided by cytoscape.js that for a View, there is a ViewModel associated with it. The ViewModel is then processed to construct a String of standard parity game format containing relevant data of current view to send to the backend.

The reason for constructing a String of the standard format is because the client provides ANTLR-generated parser classes to parse the standard format into Java objects that can be manipulated by algorithms.

From Model to ViewModel
The backend should return the computed step(s) to the frontend, it should contain at least the status of relevant nodes, and the data format should be easily parsed into a ViewModel. Two important points to note are:

- The data format should be able to describe the way a node is visualized. The slides provided by the client shows there are two dimensions of visualization of a node: the color of a node, and the effect of the color (highlighted/shaded). In the meetings with the client, it is also specified by the client that, in a snapshot of the game, one node may have several different colors, each represents a value of an label; and it's proposed by the developers that an label may be displayed as text, in case too many colors are dazzling.
- The data format should be flexible enough to accommodate the different number of labels that a node can have, as they differ per algorithm.

The detailed design of the protocol from Model to ViewModel is in the later chapter.

3.3.3. Frontend

Jakob Nielsen's 10 usability heuristics (Nielsen, 2020) are the most widely used principles in UI design, so it is adopted as the guideline for frontend design of the product. The following are the most relevant ones and the way these heuristics are associated with the product.

3.3.3.1. Visibility of system status

> *"The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time."*

The most important system status in a snapshot is the status of each node. It should be clearly visible to the user, and the user should be able to dynamically change what information to see per node and get immediate visual feedback.

The secondary system status is how far the user has proceeded in all the steps of an algorithm. This should be visually indicated by a progress bar.

3.3.3.2. Match between system and the real world

> *"The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time."*

The items in the frontend should have displayed names that are self-explanatory, or be consistent with the user. For example, since the client calls the properties of a node as "labels", they should be displayed bearing the name "labels".

The items should be placed in an order that suits the workflow of the client. Specifically, the user would like to choose an algorithm first, then import graph from file or construct graph by hand, then run the algorithm and examine the visuals, make adjustments and run the algorithm again. Items that are clicked first should appear early, and the items that are clicked consecutively should be close to each other.

3.3.3.3. User control and freedom

> *"Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without having to go through an extended process."*

There should be undo/redo in editing graphs, such that, for example when the user deleted nodes by accident, there is a quick way to recover.

There are two occasions that the user may want to clear content on the screen: one is to stop the current algorithm to edit the graph, the other is to go to an empty screen to start from scratch again. Correspondingly, for the first case, there should be a "stop" option to clear the visualizations related to the current algorithm, such as node status and progress bar status, but leave the parity game itself as a basis for the user to alter; for the second case, there should be a "clear" option to clear everything in sight, including the parity game itself, this would leave a blank canvas for the user to import game or to construct game from scratch.

3.3.3.4. Error prevention & recovery

> *"Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution."*
>
> *"...(to) prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action."*

There are several ways that could spawn an error: Is the imported file a legal parity game file? Is the game constructed by the user legal when the user chooses to run the algorithm? What if the user selected an algorithm but forget to choose how each label is displayed? Is the parity game legal to be exported? For the error-prone conditions, there should be a pre- sanity check. In some situations where precheck is not convenient or not exhaustive, there should also be after-check. The feedback to the user should be in human language, rather than machine code.

3.3.3.5. Aesthetic and minimalist design

> *"Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility."*

The aim of this project is to automate laborious actions in parity game algorithm research, implicitly it should also minimize the distractions to let the researcher focus on research itself. The options in the frontend could be categorized as "frequently used" or "less frequently used", and the "frequently used" category should be placed at the most convenient position.

4. A detailed design choices with justifications

4.1. Labels of a node

As mentioned before, a node can have a different number of labels per algorithm. The data structure in the backend and the visualization options in the frontend should be able to accommodate different labels across different algorithms.

In cytoscape.js there is a natural solution for label visualization: compound node. In a compound node, there is a child node and one or many parent node(s). A parent node is a slightly bigger node surrounding the child node. And the parent node can also have a parent node. In the product, parent nodes are used to display labels of a node.

In this section, design choices about labels are discussed.

4.1.1. Overall design of label

4.1.1.1. labels can displayed as either text or color

A label could be displayed either as a text label or a color label. We provide this flexibility to enable the users to customize how a label is displayed. In the case where a label only have a handful of possible values, a label can be displayed as color, for example, different shades of blue color; however, in the case where a label have many possible values, it is better to display it as a text, as different shades of a color is hard to distinguish by eye.

If a label is selected to be displayed, text labels will be displayed on the graph as lines of text, while color labels will be displayed on the graph as background colors of the corresponding nodes.

4.1.1.2. Color effect of the closest parent of a node

The slides provided by the client shows that a node's background color can have different strengths: it can be highlighted to stress that the node is currently being focused or the color is set for sure; or it can be shaded to represent the node is removed from the game or this is a freezed node. For this reason, a label with the name "effect" is designed, its value is either Highlight, Shade, or Neutral.

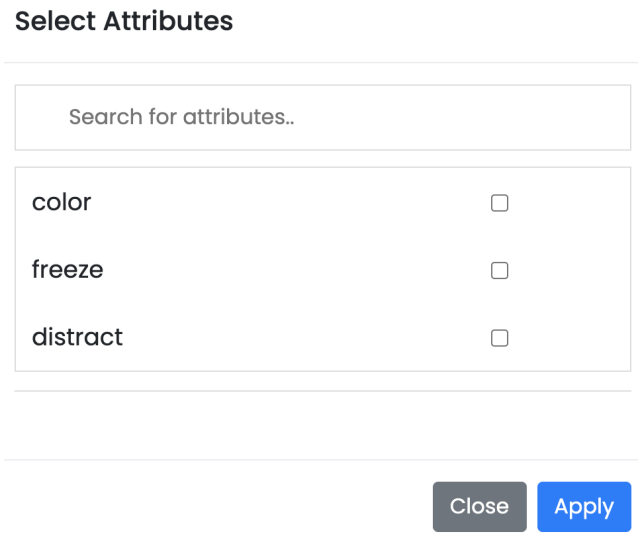
Note that the color effect can be only applied to the visualization of a node's closest parent node, if that parent node is used to display a color label. This is because for more outer parent nodes, the visible portion is smaller than the innermost parent node.

4.1.2. Visualization of label

4.1.2.1. Fetch the labels of selected algorithm

Different algorithms could have different labels associated with a node, which may result in different graphical effects displayed on the graph. Therefore, to accommodate and keep a list of the labels of customized algorithms, a request will be sent to the server asking for the labels of the algorithm when an algorithm is chosen at the frontend.

4.1.2.2. Display of color label



Complicated algorithms could have over 100 labels, which are difficult to be visualized and most of them might be unnecessary to be visualized at the same time.
When there are too many labels, the users can choose to either scroll down to select the labels or to search to select the labels which they would like to be visualized. (Note: based on client's preference, at most three labels could be selected and visualized at the same time.)
In addition, considering that users might want to view and change selected labels, each selected label will be listed at the bottom of the pop-up panel, and there is a "close" button at the rightmost of each selected label.

4.1.2.3. Display of text label

Text label is always displayed as the label of the outermost parent node.
We can only set the label of a node by assigning a value to the label node.style().label, if we want to make the label sticky to the node. However, if we add a label to the innermost node, the parent node will be stretched automatically to a rectangle to contain the text content. Therefore, to make the graph look better, we will always add a node as the outermost parent node if there is a text label and make the multiple text labels displayed in multiline as the label of that node.

4.2. Frontend

4.2.1. Collapsible Sidebar

To improve the overall UI experience, a collapsible sidebar is added to separate the less often used functionalities such as import/export game from the often used functionalities such as add node/edge, toggle ownership and turn on/off auto-layout.

4.2.2. List of steps

One of the user's requirements is that the user is able to see at which step the algorithm is running and the user can jump to any step he/she chooses to.

4.2.3. Button names and colors

To make the functionalities more intuitive, we made three buttons called **run, stop** and **clear** having three functionalities: run the algorithm, stop the algorithm and clear everything including the graph.

We made the background colors of **run** and **stop** buttons the same as theme color, while the **clear** button is in a red color. This is because the **clear** button is an irreversible action, we set its background color to red to warn the user that this extreme action may be dangerous.

4.2.4. Sequence of options in the collapsible sidebar

According to the client, he would like to select an algorithm and choose labels first. Then he would like to create a parity game by hand or import an existing parity game. Finally, he would like to export the game/solution after seeing the steps of an algorithm. Therefore, we set the "Select Algorithm" as the first option under "Parity Game", which is an expandable list, the "import" as the second option, the "export game" and the "export solution" as the third and fourth options respectively.

4.3. Backend

4.3.1. The data format to describe a node / a game

When solving a game, there can be various labels associated with a node. In designing how to return this information to the frontend, an initial thought is to make a *Vertex* object as *XMLRootElement* and construct a list of *Vertex* that will be serialized to *JSON* and returned to the frontend. However, this approach won't work because the possible labels of a node differ per algorithm, so the data returned to the frontend can't be based on a static template.

An alternative is to use a HashMap from String to String to store the labels of a node, including id, winner, strategy, labels about visualization, and other labels specific to each algorithm. And

such HashMap can be naturally serialized to JSON. This way, the flexibility is provided to the user to associate any number of labels to a node.

The relevant class is *GameStatus.java*, it's an alias of *HashMap<Integer, HashMap<String, String>>* where the key is a node's id, and the value is a node's various labels.

4.3.2. The choice of birdview game status

Initially, the game status is described in an incremental fashion: the next step makes some modifications based on the previous step. But this is not a good idea. For one thing, it makes stepping backward more inconvenient to implement. For another, it's uncertain if it is convenient for every algorithm to implement steps in such a style. Lastly, if the user wants to jump to step x, it's not convenient to compute incremental steps from the beginning up to step x every time.

The last reason is pretty realistic, consider the scenario: suppose an algorithm has a large number of steps on a graph (eg.100+) and the user just wants to have a quick look at the result, clicking "next step" 100+ times is not user friendly. Another example would be in tuning how the graph is displayed, the user made some changes only in the last few steps in the backend, then after rerun the algorithm, the user might want to directly jump to those last few steps to see the visual effects.

So we have chosen a "birdview" style of game status, which means every step we return contains complete information of the game at that step, rather than the incremental information based on the previous step.

4.3.3. Return steps to the frontend in one shot

When the user starts an algorithm in the frontend, the server can return each step upon the user's request (eg. user hitting the "next step" button), alternatively, the server can compute and return all the steps in one shot. The second option is chosen for two reasons:

The first reason stems from the fact that this is a webapp, every session of communication between the client and the server has some overhead cost. We want to minimize the overhead of communication to optimize the user experience.

More importantly, the user might not always want to step through every single step as mentioned above. We want to provide the user with the flexibility to jump to any step at any time, so after starting an algorithm in the frontend, all the steps should be available to the user.

5. Testing

The test plan consists of two parts: the unit testing and the integration testing. Specifically, in the unit testing, each operation in the frontend and the backend is tested independently and the expected result is checked. In the integration testing, operations that involve both the frontend and the backend are tested and the expected results are checked. For the integration testing, the Big Bang approach is adopted to test the entire system in one go.

The detailed test plan and the results are given in Appendix II.

6. Overview of the source code

6.1. Frontend

Here we look at the .html and .js files in the source code.

index.html
This file is the homepage.

add_edge.js
This file contains the setting of the extension that allows the user to add edges through drag and drop.

add_node.js
This file contains functions related to the functionality of adding nodes to the core graph.

auto_organize.js
This file contains the setting of the extension that allows the user to trigger the auto-layout of the graph through drag and drop. In addition, it contains the function that is used to turn on/off the functionality of auto-layout.

layout.js
This file contains functions related to auto-layout triggering.

crud.js
This file contains the functions related to CRUD services.

export.js
This file contains functions related to the functionality of exporting game/solution.

graph.js
This file contains basic style settings of the core graph (node/edge/edge handler).

import.js
This file contains functions related to the functionality of importing a game.

priority_label.js
This file contains functions related to setting the priority of the nodes.

modal_effect.js
This file contains functions related to the animation effects of the pop-up labels modal.

onload.js

This file contains functions that need to be called before the page is loaded (i.e. at the start of the program).

ownership.js
This file contains functions related to the functionality of toggling nodes' ownership.

panel.js
This file contains basic style settings about the priority and even/odd nodes in the sidebar and functions related to the functionality to add nodes by drag and drop.

select_labels.js
This file contains functions related to the functionalities of search/add/remove labels functions in the pop-up labels modal.

sidebar.js
This file contains functions related to functionalities: collapse/expand sidebar, highlight selected algorithm and post algorithm name to get its labels.

steps.js
This file contains functions related to the functionalities of step forward/backward and slider.

undo_redo.js
This file contains functions related to the functionality of group copy and paste and undo/redo functionality for deleted nodes/edges.

6.2. Backend

Here we look at the overview of Java packages, in an order of how a command from the frontend is executed.

6.2.1. Package resources

The first classes executing the command from the frontend are in this package. This package contains files that are receiving the HTTP request to certain URLs and mapping the request to corresponding methods. They are:

AlgorithmResource.java	This class is used for telling the frontend the currently available algorithms in the backend so that the frontend can display the algorithm names for the user to choose from; it can also fetch the labels of an algorithm when it's selected.
GameResource.java	This class is receiving the data about the game and the algorithm of choice when the user runs an algorithm. It then calls other classes to create an

	algorithm object, parse and solve the received game, and return steps to the frontend.

6.2.2. Package algorithms

This package contains the classes related to algorithms. They will be invoked when returning the labels or when solving a game.

Algorithm.java	The interface of all algorithms.
AlgorithmFactory.java	The class helps to create an Algorithm object.
DFI.java, PriorityPromotion.java, Zielonka.java	The built-in algorithms.

6.2.3. Package control

Solver.java	Instructed by GameResource.java, this class calls other classes to parse the game posted from the frontend, solve the game, and return the steps to the frontend.

6.2.4. Package parser

This package contains classes to parse the game from text format to Java object.

PGLexer.java, PGListener.java PGParser.java PGVisitor.java	These are classes generated by ANTLR based on the language defined in modelGame.PG.g4, they are used to parse a game of text format into Java objects that the algorithm can manipulate.

6.2.5. Package modelGame

The game is modeled by the following classes:

PG.g4	The language describing the standard parity game format. Used to generate the classes in the *parser* package.
Game.java	The game is modeled as a collection of vertices, and the successor/predecessor map of each vertex.

Vertex.java	The vertex is modeled to only maintain its (common) labels, that is id, priority, owner, and an optional label. Other labels that are specific per algorithm are defined and stored in each algorithm class.

6.2.6. Package modelStep

This package contains classes relevant to the steps returned to the frontend.

Label.java	The model for a label of an Algorithm class. An algorithm can have several labels, each label can be displayed as either “text” or “color”. If an label is going to be displayed as color, then every possible value of this label must be specified in the Algorithm class (ie. it should be specific only to the algorithm, not to a specific parity game). If a label is going to be displayed as text, then there’s no such need.
Effect.java	An Enum class of 3 possible effects for the closet parent of a node, if it’s displayed as “color”. Then the user can specify the effect of the color as either highlight, shade, or neutral (remove color). This can be useful for example, to display “frozen” nodes with color with shade effect.
GameStatus.java	As mentioned in the “Detailed design - backend” section, this class is used to model a snapshot of the game.
Step.java	This class consists of two GameStatus class variables, one for all nodes’ status and the other for the status of updated nodes; and a message to display.

6.2.7. Package test

This package contains a Benchmark algorithm (that is actually the DFI algorithm given to us by our client, Tom van Dijk) to test the correctness of the customized algorithms. The idea is customized algorithms can be tested locally without launching the webapp.
One thing to note is that failing a couple of tests does not necessarily mean the customized algorithm is wrong, because in some cases one parity game can have multiple solutions.

Benchmark.java	A DFI algorithm given by our client. Its correctness is assured.
DFITest.java *PriorityPromotionTest.java* *ZielonkaTest.java*	JUnit tests for 3 built-in algorithms.

7. User manual

This section contains information about compatibility and installation, the frontend usage, and the instructions in adding customized algorithms.

7.1. Compatibilities and Installation

7.1.1. Backend compatibility

This project is developed under JDK 11, Tomcat 9.0.21. It is recommended that users adopt the same environment. Other dependencies are specified in the pom.xml file.

7.1.2. Frontend compatibility

There is a known issue relating to setting the priority of a node/ a group of nodes. The minimal version of common browsers are listed below.

Browser	Minimal version
Chrome	51
Edge	79
Firefox	38
Safari	10

The project is developed and tested extensively under Chrome 89 and Firefox 85. It has been made sure there is no problem associated with these two browsers of the version.

7.1.3. Installation

It is proposed by the client that the project is delivered as a JAR file with Tomcat embedded. However, in the process we have encountered compatibility errors that we did not find clear and sufficient information to resolve.

An alternative way would be to install Maven and Tomcat, and deploy the web application locally. The detailed steps are as follows:

Step 1. Install Tomcat and Maven:

1. download Tomcat 9.0.21 to a local directory.
    a. For **Mac**: apache-tomcat-9.0.21.zip
    b. For **Windows-x64**, apache-tomcat-9.0.21-windows-x64.zip
    c. For **Windows-x86**, apache-tomcat-9.0.21-windows-x86.zip

2. install Maven 3.8.1. (if you already have Maven installed, skip this step)

Step 2. Create a user in Apache Tomcat:

1. navigate to the root directory of Tomcat
2. open the file **/conf/tomcat-users.xml**
3. add the following code snippet to the file, inside the `<tomcat-users>` tag:

```
<role rolename="manager"/>
<role rolename="admin"/>
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin"
roles="admin,manager,manager-gui,manager-script"/>
```

Step 3. Create a user in Apache Maven:

1. navigate to the root directory of Maven
2. open the file **/libexec/conf/settings.xml** (on Mac OSX) or **/conf/settings.xml** (on Windows)
3. add the following code snippet to the file, inside the `<servers>` tag:

```
<server>
   <id>TomcatServer</id>
   <username>admin</username>
   <password>admin</password>
</server>
```

Step 4. Start Tomcat:

1. open the terminal and navigate to the root directory of Tomcat

For **Mac:**

2. run `chmod +x catalina.sh` and `chmod +x startup.sh` to make the file **catalina.sh** and **startup.sh** executable
3. run `bin/startup.sh` to start Tomcat

For **Windows:**

We execute the following to start the Apache Tomcat:

```
bin\startup.bat
```

Step 5. Deploy the application:

1. open the terminal and navigate to the root directory of the application
2. run `mvn tomcat:deploy`
3. check on http://localhost:8080/manager/html/ and if the deployment was successful, the username and password are both “admin” as set in Step 2. There will be a column like this:

/pandavis	*None specified*	Parity Game	true

4. go to http://localhost:8080/pandavis to use the application

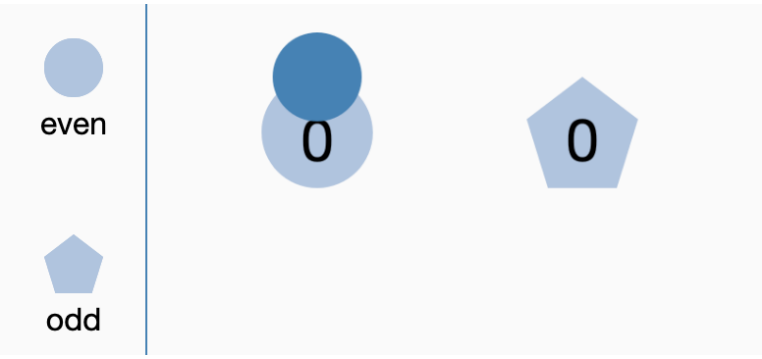
(Note: Everytime a change is made to the backend classes, to ensure the application will be updated, remember to clear the cache in the browser and the application needs to be re-deployed: open the terminal at the root directory of the application and run `mvn tomcat:redeploy`)

7.2. Frontend Usage

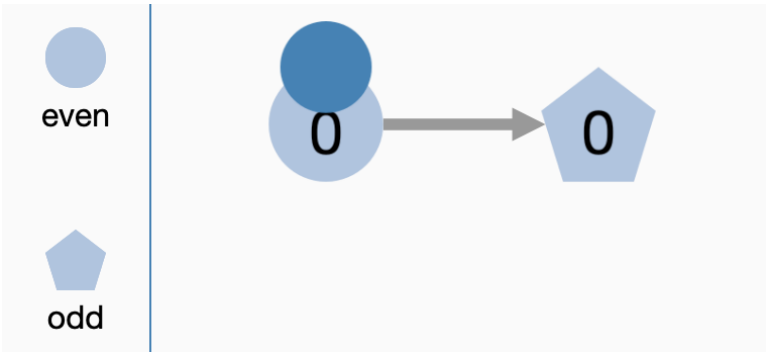
The following content is also available in a live demonstration video.

7.2.1. Add Parity Game

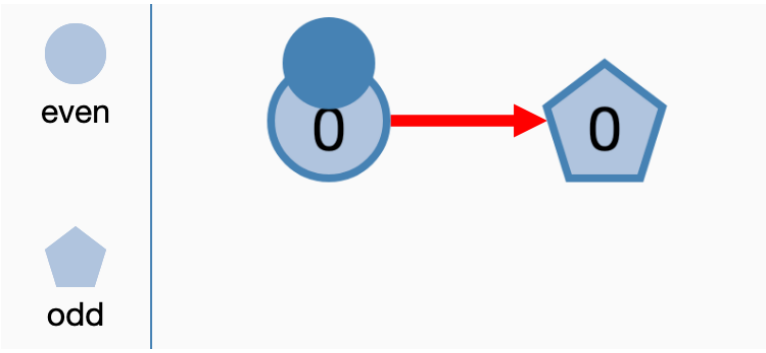
7.2.1.1. Create parity game by hand



1. Drag the even/odd node from the static sidebar to the core graph area to add an even/odd node to the core graph with default priority 0.

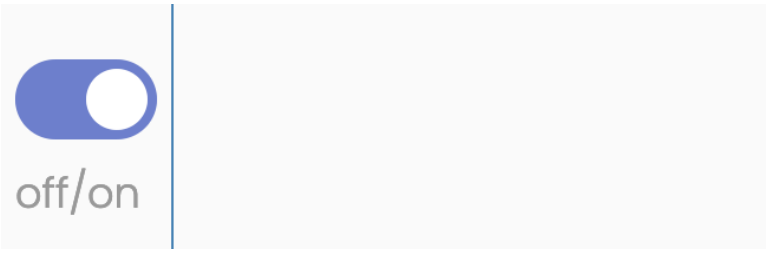


2. When the mouse is over a node, an edge handler (i.e. the blue cycle on the top of the node) will appear.

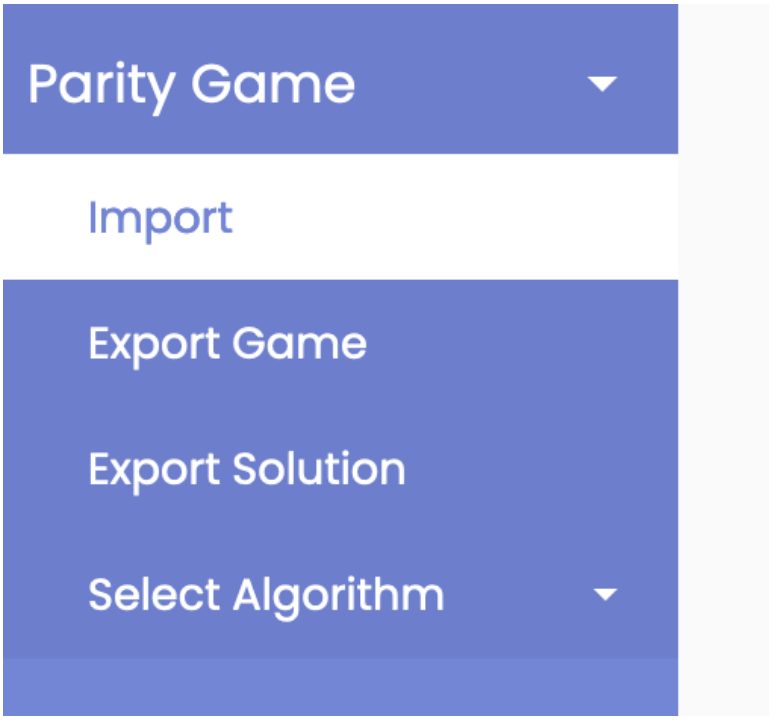


3. Drag the edge handler from this node to another node to create a directed edge.

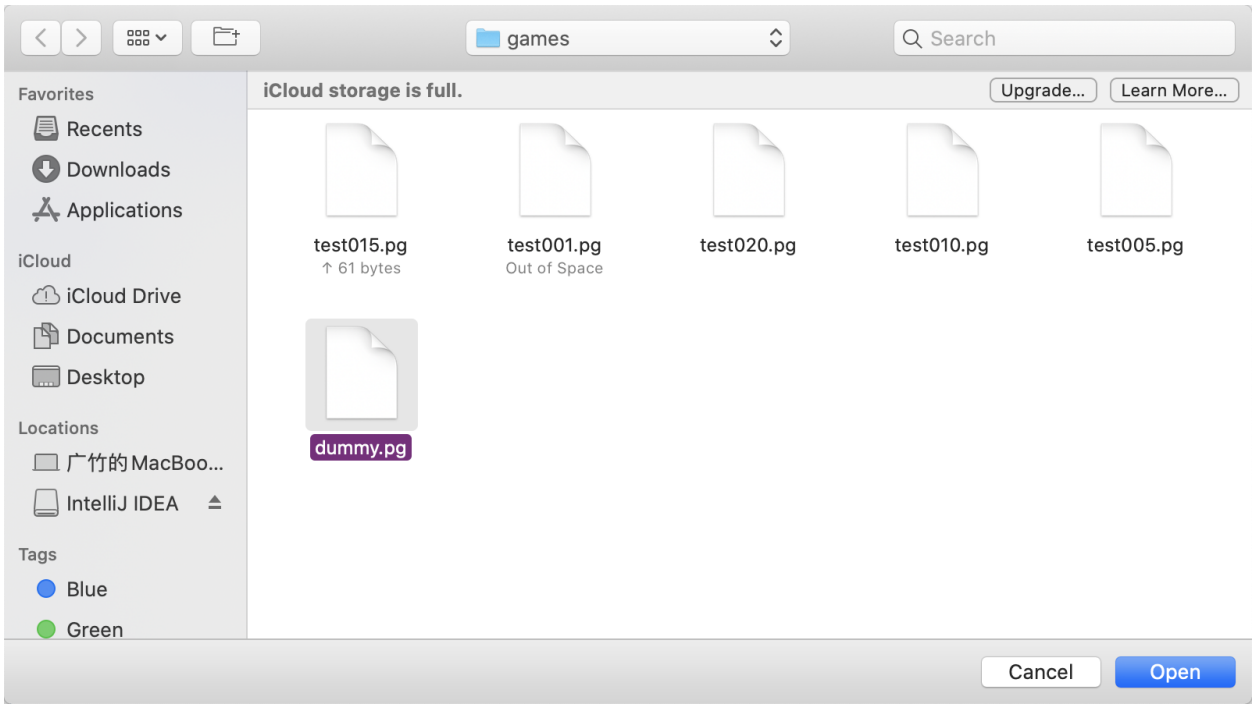
7.2.1.2. Import parity game from local .pg file



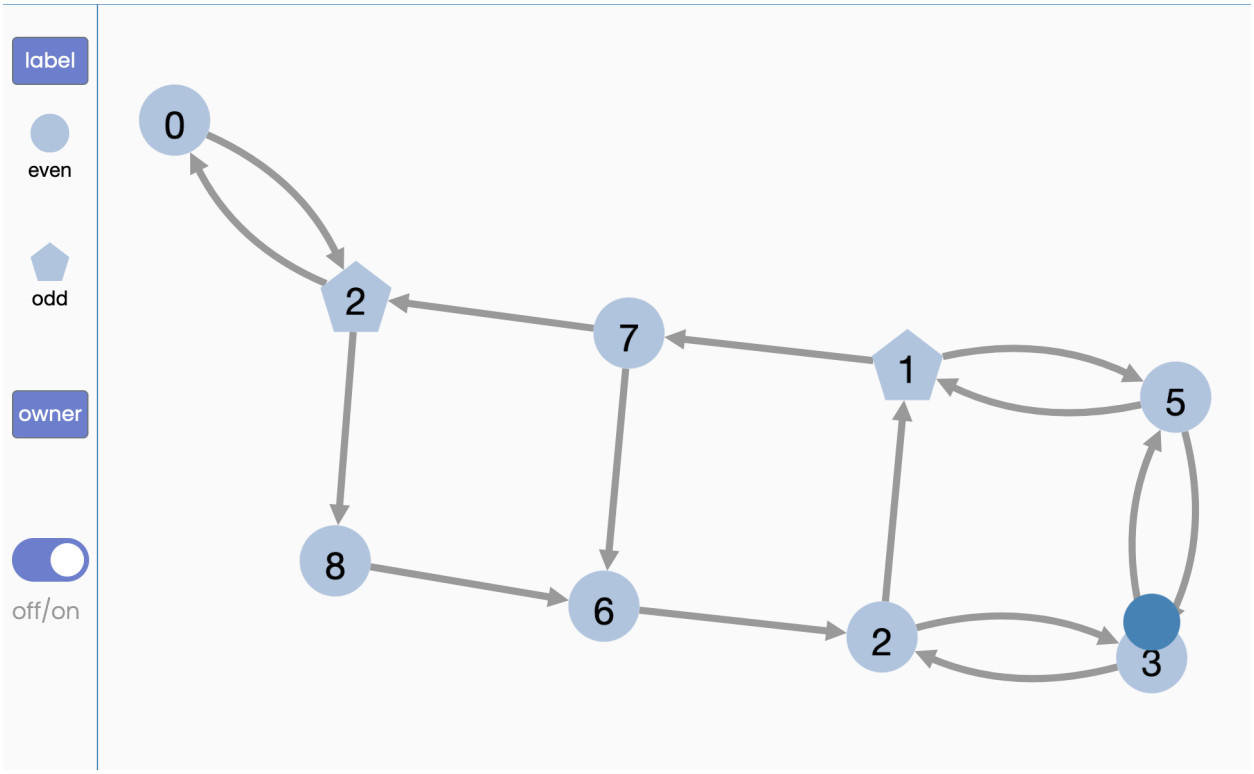
1. Turn on auto-layout (skip if auto-layout is already on).



2. Go to **Parity Game** in the collapsible sidebar, click on the **Import** option.



3. Select and open a .pg file from a local directory.



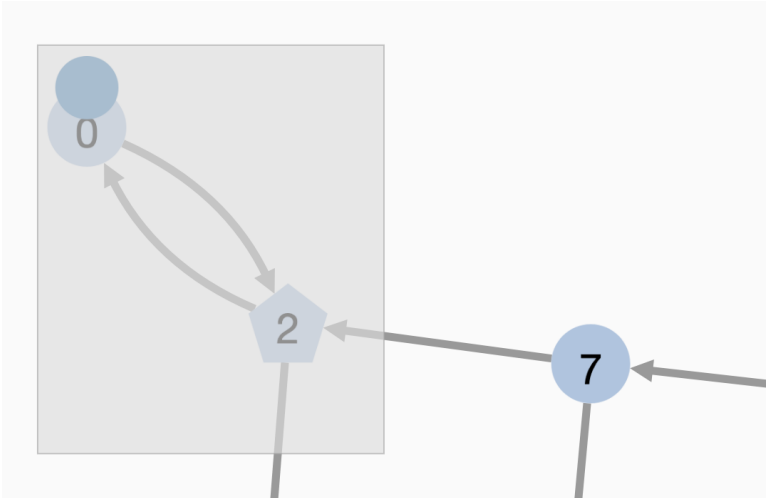
7.2.2. Configure Parity Game

7.2.2.1. Add node/edge

These operations are covered in the above section.

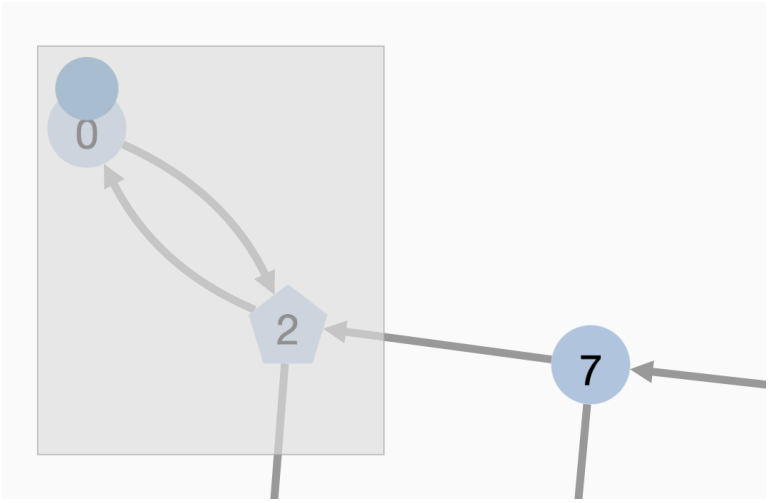
1. add node.
2. add edge.

7.2.2.2. Set priority



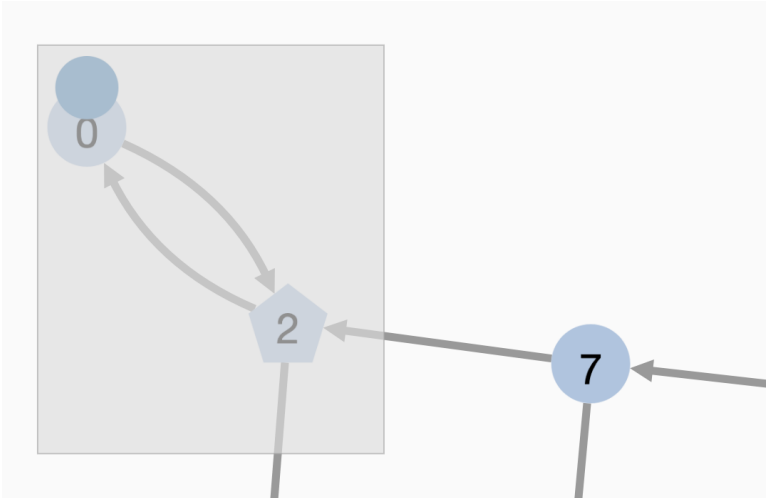
1. Press **ctrl** (for windows)**/command** (for mac) key and select one or more node(s) (click to select/box select).
2. Press **delete** (for mac)/**backspace** (for windows) key to delete the original priorities of the selected node(s).
3. Press **number keys** to input new priorities for the selected node(s).
4. Click the blank area to save.

7.2.2.3. Increment/Decrement priority

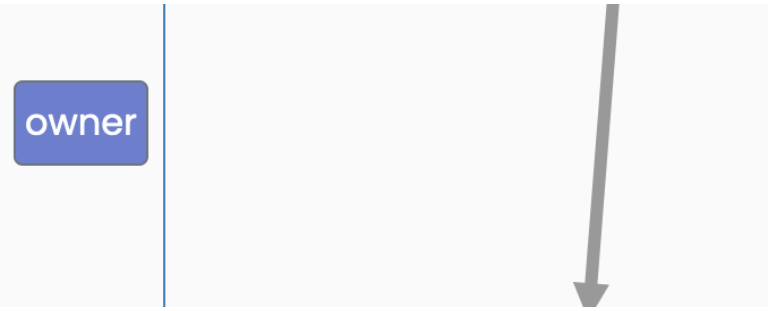


1. Press **ctrl** (for windows)**/command** (for mac) key and select one or more node(s) (click to select/box select).
2. Press **up** key to increment the priorities of the selected node(s).
3. Press **down** key to decrement the priorities of the selected node(s).
4. Click the blank area to save.

7.2.2.4. Toggle ownership

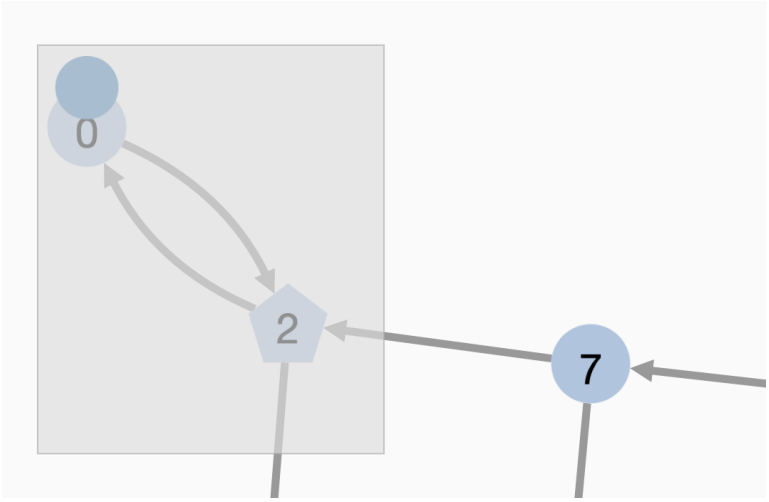


1. Press **ctrl** (for windows)**/command** (for mac) key and select one or more node(s) (click to select/box select).



2. Click on the **owner** button to toggle the ownerships of the selected node(s).

7.2.2.5. Delete node/edge



1. Press **ctrl** (for windows)**/command** (for mac) key and select one or more node(s) (click to select/box select).
2. Press **delete/fn + delete** keys to delete the selected node(s).
3. Press **ctrl + z** keys to undo deletion**.**
4. Press **ctrl + y** keys to redo deletion.

7.2.2.6. Copy/paste subgraph

(NOTE: to avoid that the pasted subgraph cannot be found, the auto-organize function should be turned on.)

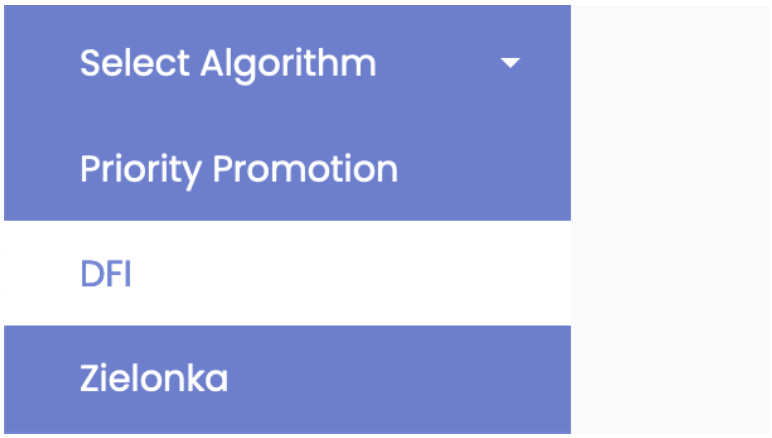
1. Press **ctrl** (for windows)**/command** (for mac) key and select one or more node(s) and edges(s) (click to select/box select).
2. While pressing **ctrl/command** key (do not release!),click the blank area
    a. The reason behind this is that cytoscape.js has no native method to set/increment/decrement a node's priority, this functionality is achieved by using a hidden input field, and the input field will be triggered whenever a node or a

group of nodes is selected. Clicking the blank area will leave the hidden input box, and then the **ctrl + v** key can be listened to.

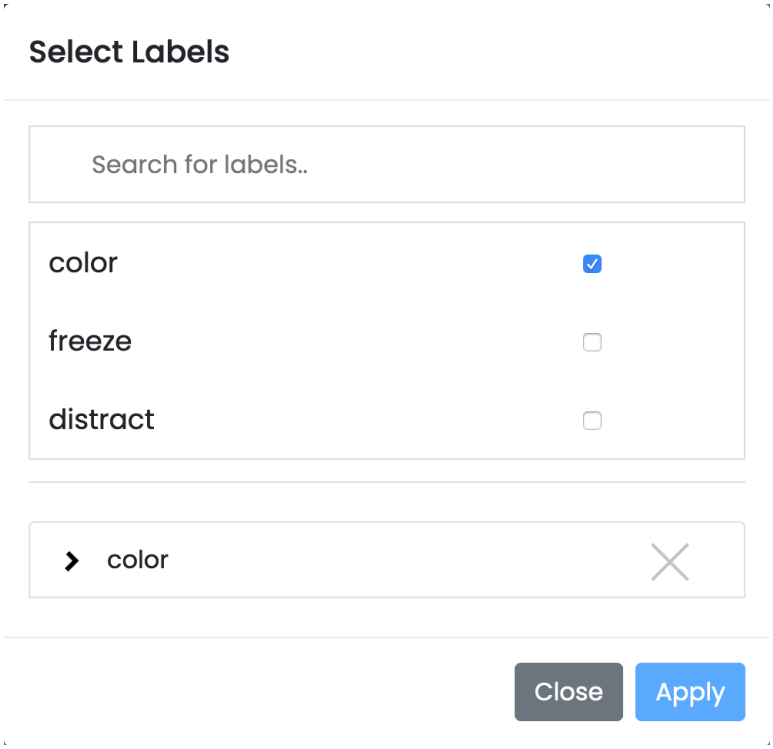
3. Press **ctrl + v** keys to paste the selected subgraph.

7.2.3. Algorithm

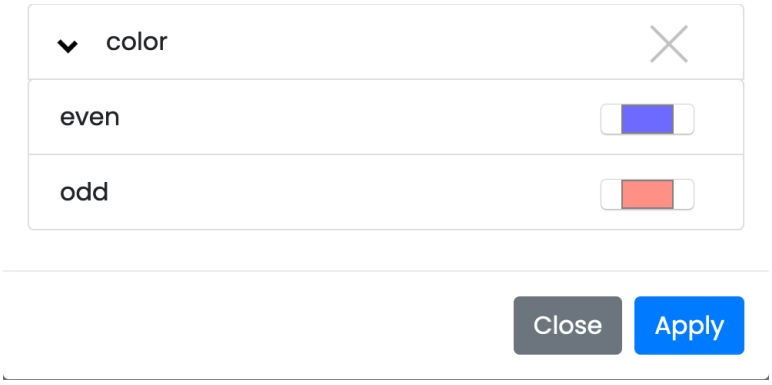
7.2.3.1. Run algorithm



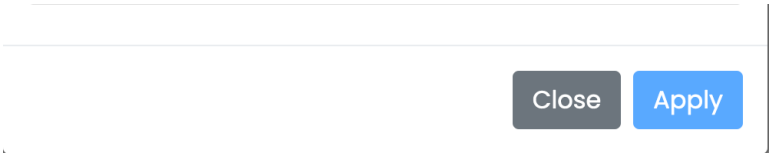
1. Go to **Parit Game -> Select Algorithm** in the collapsible sidebar, click to select an algorithm (skip if an algorithm is already selected).



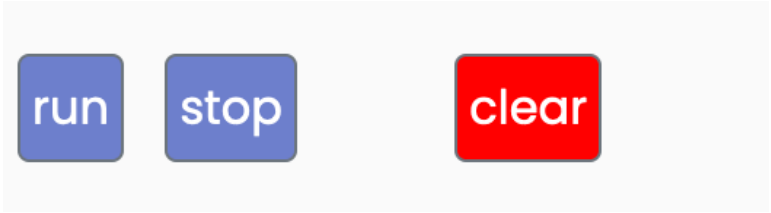
2. In the pop-up modal, select at most 3 labels to be visualized.



3. Choose a color for each value of each label of type color.

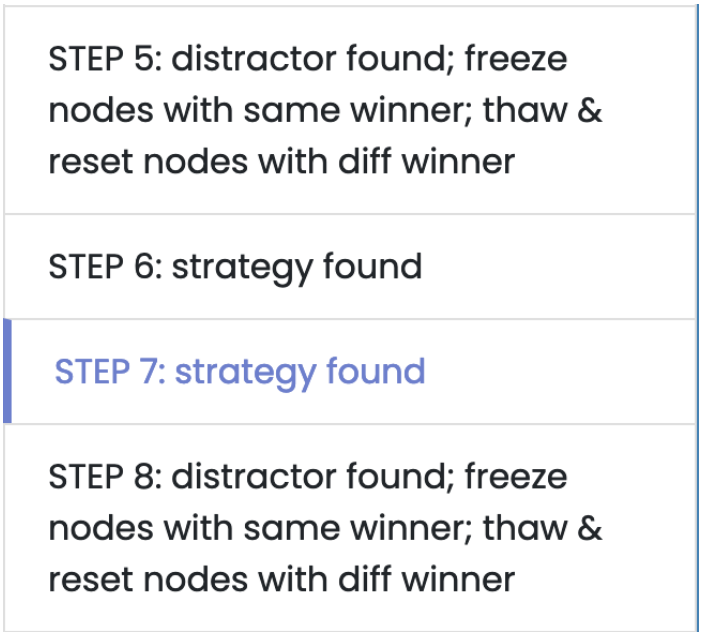


4. Click on the **Apply** button to apply the label setting.

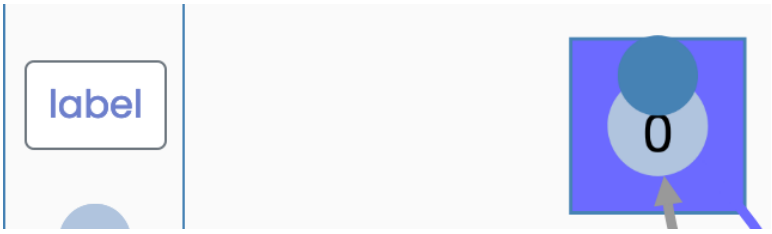


5. Click on the **run** button to run the algorithm on the current core graph.

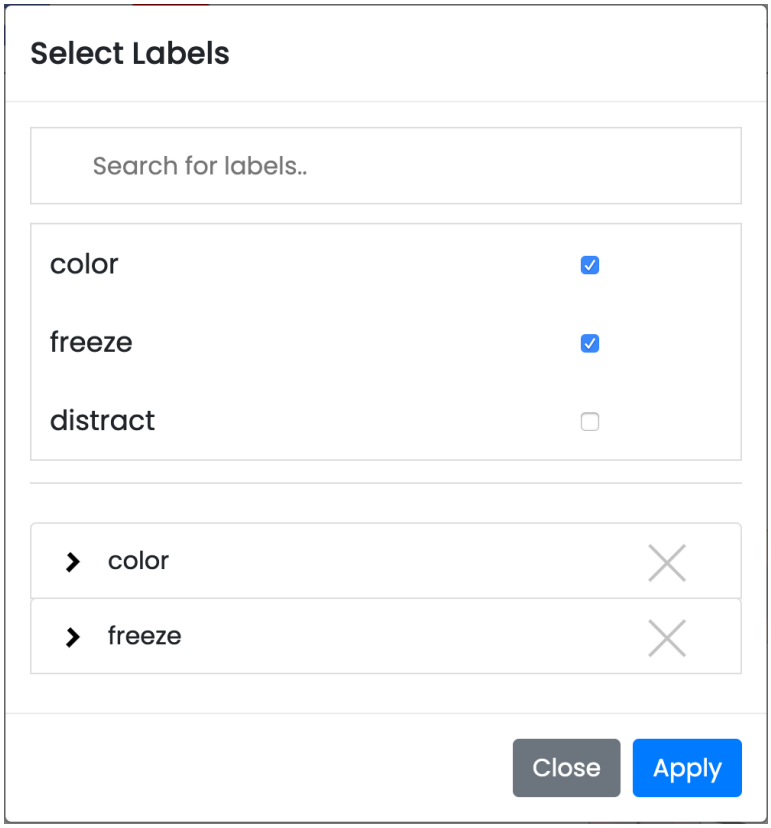
6. Click on the **>> / <<** button to step forward/backward or use the slider to visualize all steps at once.



7. Click any step in the right list to jump to that step.

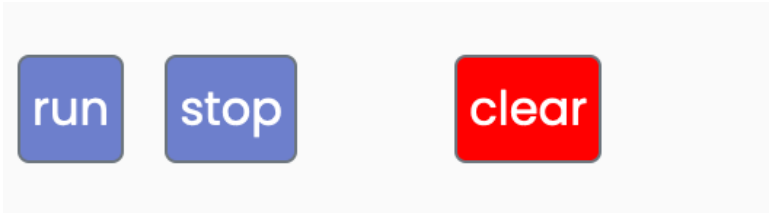


8. Click on the **label** button to open the label setting modal.



9. Modify the label setting and click on the **Apply** button to apply the setting.
10. Continue visualizing the steps.

7.2.3.2. Stop algorithm



1. Click the **stop** button to stop the algorithm, which will reset the steps.
2. After making several changes to the graph the user can re-run the selected algorithm on the modified core graph immediately with the previously saved label setting by clicking the **run** button, if the user does not want to change the labels to be displayed.

7.2.4. Export Game/Solution

1. Go to **Parity Game,** click on the **Export Game** option.
2. Choose to save the .pg file in the pop-up window.

7.3. Add customized algorithm

7.3.1. Procedure

1. When writing a customized algorithm, you can use the classes in the modelGame package to model the game. Alternatively, you may want to use other data structures to represent a game, this way you need to update the *parse* method in *Solver.java* to parse the game into your desired data structure.
2. The customized algorithm should implement *Algorithm.java* interface for *AlgorithmFactory.java* to work correctly later on.
3. The inherited methods from *Algorithm.java:*
    a. *solve, getWinner, getStrategy* are self-explanatory
    b. *getSteps* returns a collection of *Step* objects. Please refer to the *"Package modelStep"* section and/or the comments in the code for detailed information.
    c. *getLabels* returns the labels specific to this algorithm. Please refer to the *"Package modelStep"* section and/or the comments in the code for detailed information.
4. (optional) After writing the algorithm, you can test its correctness by writing a JUnit test class in the test package.
5. Please go to *AlgorithmFactory.java* of the "algorithm" package, and add a clause accommodating your customized algorithm for each of the methods.
    a. *getAlgorithm* method will enable the creation of the Algorithm object in the backend.

b. *getLabel* method will pop up a modal with the labels of this algorithm when it is selected in the frontend. The modal is for you to choose which ones to display among all the labels, and what color should represent what value.

6. Please go to *AlgorithmResource.java* of the “resources” package, and add the name of the algorithm in *getAlgorithms* method, so that when the webapp is launched, the frontend knows that this customized algorithm is available, and it will be displayed under “select algorithm” in the collapsible sidebar.

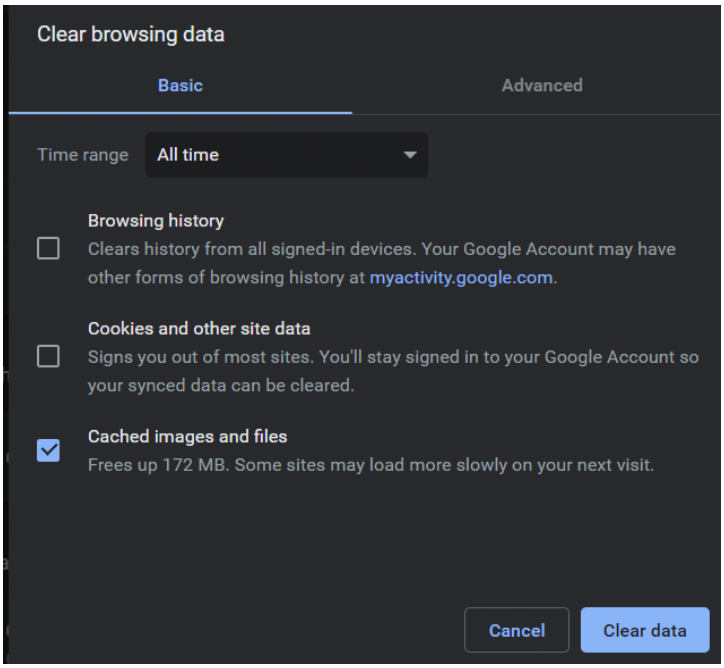
7.3.2. Note

There are two issues about caches when modifying the backend:

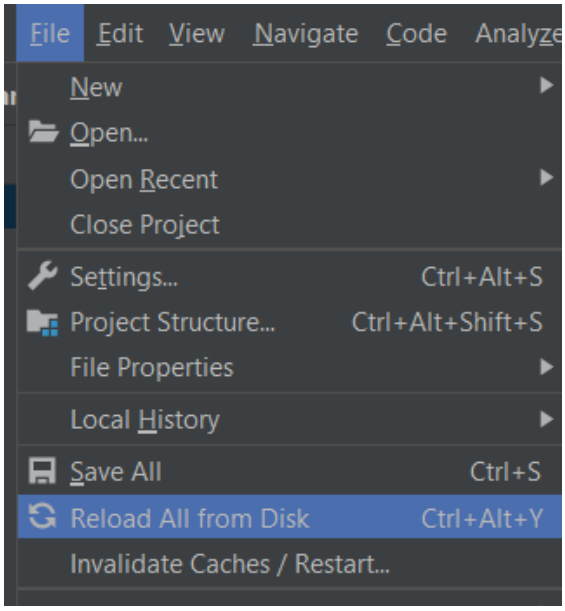
The first is the cache in the browser. After modifying the backend code and re-deploy, the launched web page may not reflect the latest changes. This is due to the browser having cached the deployed file. In order to make sure that the changes are reflected in the browser, you need to clear the cached files.

In Chrome this is done by:

1. Clicking the three dot in the top-right corner
2. Choose “Settings”
3. Go to “Privacy and Security” - “Clear Browsing Data”
4. Only the last option, “Cached images and files” needs to be ticked.

a. 

The second issue happens less frequently. The IDE may cache the files such that the change won’t be reflected. The solution is to clear the cache. In Intellij this can be done by clicking File in the top left corner, and choosing “Reload All from Dist” and/or “Invalidate Caches / Restart”.


File
Edit
View
Navigate
Code
Analyze
New
Open...
Open Recent
Close Project
Settings...
Ctrl+Alt+S
Project Structure...
Ctrl+Alt+Shift+S
File Properties
Local History
Save All
Ctrl+S
Reload All from Disk
Ctrl+Alt+Y
Invalidate Caches / Restart...

8. Individual contribution

8.1. Weiting

- Contribute to design and implementation of the overall GUI
- Contribute to edit parity game: add/remove edges, toggle ownership
- Redo/undo deletion
- Contribute to Copy/Paste subgraph
- Contribute to auto-organization.
- Contribute to parts of control the algorithm:
    - design data structure for algorithm labels
    - choose displayed labels and choose color for the values
    - run an algorithm
    - display step messages
    - jump to a specific step
    - use slider to change steps
    - stop algorithm & clear everything
- Peer review 1 and 3

8.2. Di

- Contribute to design and implementation of the overall GUI
- Contribute to edit parity game: add/remove nodes, toggle ownership
- Contribute to auto-organization
- Select an algorithm in the frontend
- Contribute to parts of control the algorithm:
    - design data structure for algorithm labels
    - pop-up modal for searching and selecting displayed labels
    - choose displayed labels and choose color for the values
    - run an algorithm
    - display step messages
    - step forward and step backward
    - stop algorithm & clear everything
- Demo video

8.3. Ruilin

- Proposed the draft requirements.
- Implementation of the backend.
- Serialize the game and post it to the backend.
- Design the data structure for steps.
- Implemented 3 built-in algorithms that can return steps to the frontend.
- The import/export functions and the set single/group priority function.

- Logistics:
    - peer review session 2
    - final presentation design and deliver
    - poster design

8.4. Group work

All developers contributed to the works in this section.

- Writing the project proposal.
- Writing the design report.
- Meeting with the client.
- Attend group meetings, this includes exchanging the understanding about the client's specific requirements, setting the goal of the next step, and communicating about bugs.

9. Future work

The most significant drawback, in our opinion, is that the webapp architecture requires deployment. Any slight change in the backend requires rebuild and redeploy, this can take about half a minute. Although the user can test an algorithm's correctness without launching the webapp, it may need some tuning to achieve the desired visual effects so the user might still need to redeploy a few times.

The optimal solution would be a desktop application which eliminates the waiting at redeployment. We did not think of it as a choice because the choice of a Javascript library naturally led us to choose webapp.

However, near the very end of the project, it is discovered that there exists a Javascript library (*Electron.js*, n.d.) that can build desktop applications. If this product made use of this library, it would be much more convenient to edit the background algorithms. The only constraint that is imposed on the user is that the user needs to implement parity game algorithms in Javascript. Due to time constraints, we did not ask if this is acceptable to the user. For the future possible work, it may be considered to build a desktop application with Electron.js.

References

Cytoscape.js. (n.d.). https://js.cytoscape.org/

Electron.js. (n.d.). https://www.electronjs.org/

Nielsen, J. (2020, Nov 15). *10 Usability Heuristics for User Interface Design*.

https://www.nngroup.com/articles/ten-usability-heuristics/

Appendix I: requirement specification

Since the only stakeholder is the researcher, in the table “I” always refers to the researcher.

category	requirement	priority	implemented?	reasonForNoImplementation
adjust the game	I want to add/remove nodes and add/remove edges on the graph.	Must	Yes	-
adjust the game	I want to set/switch ownership of nodes.	Must	Yes	-
adjust the game	I want to change the priority of nodes.	Must	Yes	-
adjust the game	I want to select a group of nodes and do copy-and-paste.	Must	Yes	-
adjust the game	I want to select a group of nodes and set their priority in one go.	Must	Yes	-
adjust the game	I want to select a group of nodes and toggle their ownership in one go.	Must	Yes	-
adjust the game	I want to have a way to easily increment/decrement the priority of a node by 1.	Must	Yes	-
adjust the game	I want to have a way to easily undo/redo the previous action.	Must	Yes	-
adjust the game	I want to have a clear button to clear all elements on the graph.	Must	Yes	-
Import/export/generate the game in GUI	I want to construct a parity game by hand in the GUI.	Must	Yes	-
Import/export/generate the game in GUI	I want to import parity games from .pg files.	Must	Yes	-
Import/export/generate the game in GUI	I want to export parity games (the game and/or its solution) to standard parity game format.	Must	Yes	-
Import/export/generate the game in GUI	I want to generate a random parity game in the GUI.(it doesn’t have too much meaning)	Could	No	It turns out to be not that necessary.
Visual aid for	I want to have a way to select which	Must	Yes	-

research	labels to display.			
Visual aid for research	I want the dynamic of the algorithm to be reflected in the GUI.	Must	Yes	-
Visual aid for research	I want the nodes and edges not overlapping with each other.	Must	Yes	
Visual aid for research	I want the graph to be auto-organized to a nice layout.	Could	Yes	-
Visual aid for research	I want to have a way to toggle on/off the auto-organize feature.	Could	Yes	-
Visual aid for research	The GUI should be able to display a node that has an edge pointing to itself.	Must	Yes	-
Visual aid for research	I want to see the performance of the algorithm by seeing how many steps it has taken.	Must	Yes	-
Visual aid for research	When a node/edge is selected, it is highlighted.	Must	Yes	-
Visual aid for research	I want to see, when a node is selected, the node and the other nodes currently in the same region being highlighted.	Should	No	Not every algorithm has the concept of "region", this requirement is generalized as the first requirement in this section.
Start/stop/pause/step control	I want to start the algorithm from GUI.	Must	Yes	-
Start/stop/pause/step control	I want to stop the algorithm. (clear steps)	Must	Yes	-
Start/stop/pause/step control	I want to forward each step of the algorithm.	Must	Yes	-
Start/stop/pause/step control	I want to backward each step of the algorithm.	Must	Yes	-
Start/stop/pause/step control	I want to have the option to forward/backward by big step or small step.	Should	No	Replaced by selecting step in the list of steps in the frontend & inserting step in algorithm in the backend as the

				user wishes. The user can define big/small steps as desired in the algorithm.
Start/stop/pause/step control	I want to have a slider to slide through the steps	Should	Yes	-
Start/stop/pause/step control	I want to have the option to set how quick the steps should run on the GUI (eg. steps/second)	Could	No	This does not make much sense when the user can forward/backward and jump anywhere
Start/stop/pause/step control	I want to pause the algorithm. (there's no auto-play of the algorithm, only manually step forward/backward, so pause does not make sense)	Could	No	This does not make much sense when there's no auto-play of the algorithm. Every step is by default displayed static.
Algorithms	I want to add customized algorithms.	Must	Yes	
Algorithms	I want the fix point algorithm (DFI) built-in to the product.	Could	Yes	
Algorithms	I want the priority promotion algorithm built-in to the product.	Could	Yes	
Algorithms	I want the Zielonka algorithm built-in to the product.	Could	Yes	
Algorithms	I want the small progress measures algorithm built-in to the product.	Could	No	There's not enough time.
Non-functional requirements	There is a user onboarding guide / quick help function.	Could	No	Replaced by video demo & user manual.
Non-functional requirements	I want the framework to be extensible for other (graph) algorithmic studies.	Should	Yes and No	As developers, we don't know what is common across different types of graph algorithm study, currently many options in the frontend are specific to parity

				games, such as import/export/set attributes. If a researcher wants to extend it, there could be a lot of work to be done in frontend.

Appendix II: test plan and test result

Unit testing

In this section every operation is singled out and tested separately.

#	Category	Description	Expected result	Test Class
1	Edit Parity Game in the GUI	I want to add/remove nodes and add/remove edges on the graph.	When a node is dragged from the static sidebar to the core graph area, a new node will be created. When the edge handler(a big circle appearing on a node when the node is hovered) is dragged, a directed edge from the source node pointing to the target node will be created. When a node/edge or a group of nodes/edge are selected and the key **Delete** (for windows) or **Fn + backspace** (for mac) are pressed, the selected nodes/edges will disappear. If a node/edge or a group of nodes/edge were deleted, when the key **ctrl + z** are pressed, the deleted node(s)/edge(s) will appear. Pressing **ctrl + y** will redo the deletion.	add_node.js add_edge.js
2	Edit Parity Game in the GUI	I want to toggle ownership of node(s).	When the **owner** button is clicked, the ownership of the selected node(s) will be toggled.	ownership.js

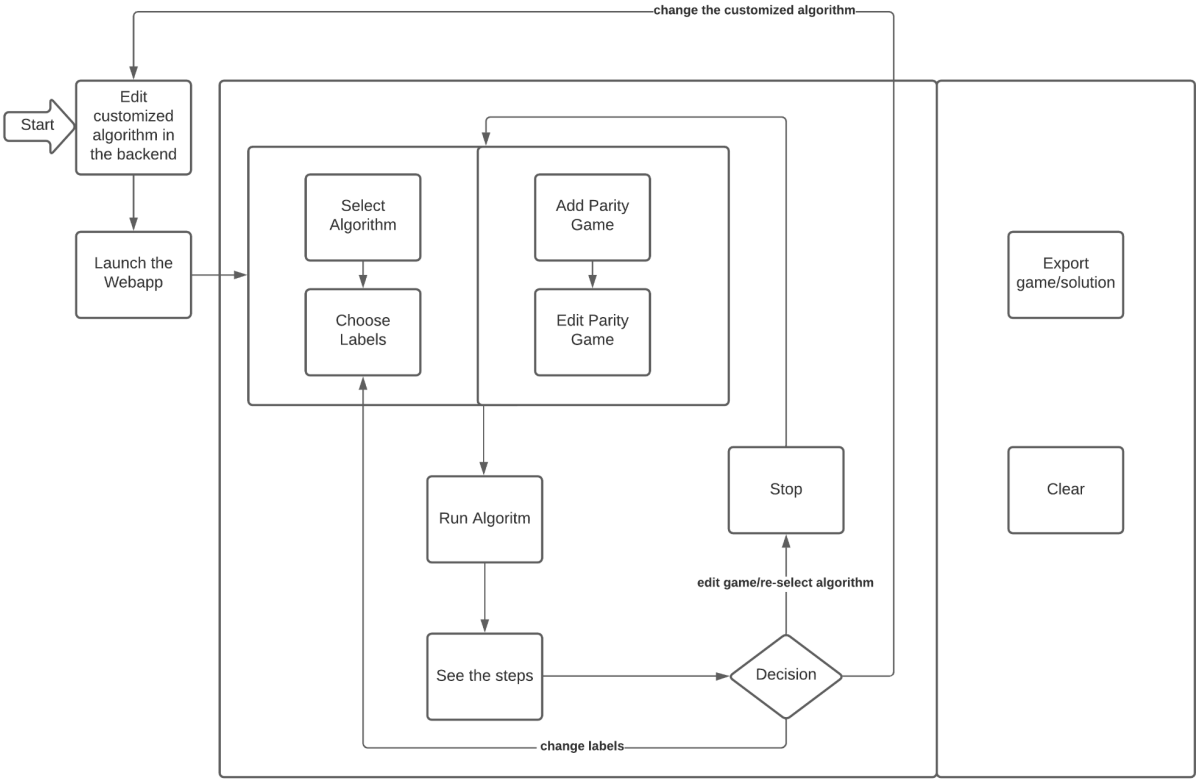
3	Edit Parity Game in the GUI	I want to set priority for node(s).	When a node or a group of nodes are selected and **delete** (for mac)/**backspace** (for windows) key are pressed, the original priorities of the selected node(s) will disappear. When a node or a group of nodes are selected and several **number** keys are pressed, new priorities for the selected node(s) will appear. When a node or a group of nodes are selected and the **Up/Down arrow** is pressed, the priority of the selected node(s) will be incremented/decremented by 1.	priority_label.js
4	Edit Parity Game in the GUI	I want to copy & paste a selected subgraph.	When a node or a group of nodes are selected, while pressing ctrl/command key, click the blank area. Then after pressing **ctrl + v**, the selected nodes/edges will be pasted.	undo_redo.js
5	Aesthetic	I want the layout of the graph to be auto-organized.	When the “auto-organize” functionality is turned on, and a node in the core graph is dragged, the nodes in the graph won’t overlap, the edges won’t tangle with each other. The position of the nodes can be adjusted automatically to be appropriate.	layout.js
6	Aesthetic	I want to turn on/off the “auto-organize” functionality.	When the “on/off” switch is toggled, the “auto-organize” functionality is turned on/off.	layout.js
7	Control Algorithm	I want to run the algorithm from the frontend.	When the “run” button is clicked, the algorithm starts.	crud.js Java package “Resources”

8	Control Algorithm	Stop the algorithm.	When the “stop” button is clicked, the algorithm stops and the stored steps at the client side is cleared.	Solver.java steps .js
9	Control Algorithm	I want to clear everything.	When the “clear” button is clicked, everything will be cleared.	steps.js
10	Control Algorithm	I want to forward each step of the algorithm.	When the “forward” button is clicked, the algorithm takes one step forward, the graph and the step log messages change accordingly.	steps.js
11	Control Algorithm	I want to backward each step of the algorithm.	When the “backward” button is clicked, the algorithm takes one step backward, the graph and the step log messages change accordingly.	steps.js
12	Control Algorithm	I want to jump to any step of the algorithm.	When any step contained in the right step list is selected, the algorithm will jump to that step. The graph and the step log messages change accordingly.	steps.js
13	Control Algorithm	I want to have a slider.	When the user slides the slider, the status of the graph changes to the step corresponding to the slider.	steps.js
14	Import / Export	I want to import parity games from standard parity game files.	When the “import” button is clicked, the user will be able to choose a file from the local directory, and the visual representation of the game will be displayed on the UI.	import.js
15	Import / Export	I want to export parity games/solutions to games to standard parity game format.	When the export button is clicked, the solution of the current parity game (if available) will be exported to a file in standard parity game format. After running an algorithm, the user can export the solutions to a file in standard parity game format.	export.js
16	Test Algorithm	I want the fix point iteration algorithm built-in to the product.	A correct DFI algorithm is implemented in the backend.	DFI.java DFITest.java

17	Test Algorithm	I want the priority promotion algorithm built-in to the product.	A correct Priority Promotion algorithm is implemented in the backend.	PriorityPromotion.java PriorityPromotionTest.java
18	Test Algorithm	I want the Zielonka algorithm built-in to the product.	A correct Zielonka algorithm is implemented in the backend.	Zielonka.java ZielonkaTest.java

Big-Bang Integration Testing

In this section we tested the whole workflow of the product, each operation can be done in varied ways, some of them will lead to success while some of them will trigger error protection messages. The following chart illustrates the flow of our testing.



Choose algorithm & game

At the beginning, the user can edit a customized algorithm in the backend. After launching the application, the user can choose to select an algorithm and choose the displayed labels for the selected algorithm or to add a parity game and edit the added parity game. Both operations need to be done before the user can proceed to start the algorithm, but their order does not matter.

After running the algorithm

The algorithm can only be started if an algorithm is selected in the collapsible sidebar, and the current graph is a legal parity game. “Legal” here means every node has at least 1 successor, and every node has a priority, otherwise there would be an alert message.
After an algorithm is run, the user can click the step message or drag the slide bar to jump to any step to see its visualization. Then the user might want to do one or more of the following actions:

- Re-choose which label(s) to see
- Edit the game or re-select an algorithm
- Edit the algorithm in the backend

If the user wants to change the label(s) being displayed, the user can click the “Label” button to configure and the changes will be applied immediately. If the user wants to edit the game or run another algorithm on the same graph, the user might want to clear all the steps and visualizations but keep the game itself, in this case the user can click the “Stop” button. If the user gained some insight from the visualization and wants to edit the algorithm in the backend, then after editing the algorithm, the user needs to relaunch the algorithm and start all over again.

Export and clear are parallel to the above

After launching the webapp, in parallel with all operations above are the export operation and the clear operation. These two operations are allowed at any stage. However, the export game/solution operation could trigger error protection mechanism in one of the following cases:

- the parity game is empty
- the parity game is invalid
- no algorithm has been run

Appendix III: weekly meeting/demo with supervisor

Week 1 - Feb 4, Thu

- Get to know each other.
- Validated proposed requirements.
- Get to know the context a bit.
    - Oink
    - Typical size of a graph
    - Parity game rules
- Get to know the constraints.
    - Platform
    - Programming language

Week 2 - Feb 11, Thu

- Get to know the context a bit more
    - Interpretation of the parity game text format
    - Typical time needed to solve a game
    - The easiest algorithm to start with
- Discussed about proposal
    - The possible risks

Week 3 - Feb 18, Thu

- Get to know the context a bit more
    - Interpretation of the parity game solution format
    - Get sample code of an algorithm, get to know the workflow
- Discussed about implementation
    - Get advice about factory pattern
    - Communicated the change of framework, because the first choice (JavaFX) doesn't work
- Discussed about proposal
    - Clarified some requirements' meaning

Week 4 - Mar 1, Mon

- Demo
    - Can add node by drag-and-drop
    - Can add node by clicking "add node" button
    - Can add edges between nodes
- Requirements

    - Get new requirements about copy and paste nodes.
    - Get updated requirements about change a node's property

Week 5 - Mar 8, Mon

- Demo
    - Can copy-and-paste nodes and edges
    - Can set priority by an external input box and a button
    - Can return dummy steps from the backend
- Get contacts of students who could give feedback
- Requirements
    - Get updated requirements about increment/decrement a node's priority
    - Get feedback from demo to highlight a node when it is selected

Week 6 - Mar 15, Mon

- Demo
    - Can set priority by keyboard input (number/up/down/delete) without an external input box and a button
    - Can import and visualize parity games
    - The layout can be automatically organized to look nice
    - Can return real steps from the backend computed by Priority Promotion
- Discussed about implementation
    - Implementing which algorithm next can help us gain more understanding of the context?
- Requirements
    - Get feedback from demo to have an option to turn on/off the auto-organize feature
    - Get updated requirements about the display of a node's label.

Week 7 - Mar 22, Mon

- Demo
    - An overhaul of the frontend to have a nicer look
    - Can display the message associated with each step
    - Can export solution/game
    - Refactored the code to accommodate different labels of different algorithms
- Discussed about project delivery
    - Get advice on deployment of the webapp

Week 8 - Mar 29, Mon

- Demo
    - Can select which labels to visualize
    - Can jump to any step by dragging slide bar and clicking step message
    - Auto-organize can be turned on/off
    - More built-in algorithm can be selected (DFI, Zielonka)

Week 9 - Apr 6, Tue

- Demo
    - Strategic edges are colored
    - User can display a label as text
    - User can select/change algorithm from GUI
- Discussed about communication within group

Week 10 - Apr 12, Mon

- Get to know the context a bit more
    - Get to know a parity game may have multiple solutions
    - Validate the workflow to guide the design of the frontend

Appendix IV: sprint plan and result

Sprint Number	Sprint Name	Deliverables	Reality
1	wk1: Kick-off	-	
2	wk2: Design and Mockup	Proposal and Planning	As expected
3	wk3-wk4: Development	Users are able to load a parity game, run an algorithm on it, and control the start/stop/pace/step of the game.	Slightly behind Mostly due to the change of the visualization library. The user can add nodes and edges.
4	wk5-wk6: Development	Users are able to construct games from GUI and see the process of running algorithms reflected on the graph.	Slightly ahead. In addition to the plan, the layout can be automatically organized.
5	wk7-wk8: Development and Test	Users are able to get visual aid (seeing the grouped nodes highlighted/toggle label of nodes), and edit customized algorithms within GUI.	Roughly as expected. Because some of the planned deliverables are based on requirements that are later invalidated.
6	wk9: Finalizing	Report and User Manual	Fixing bugs and improving the UI.
	wk10	(not planned)	Improving the UI, writing reports.